REMARKS

Claim rejections under 35 USC 102

Claims 1-15 and 16-20 have been rejected under 35 USC 102(e) as being anticipated by Morshed (6,760,903). Claims 1, 14, and 18 are independent claims, from which the remaining pending claims ultimately depend. Applicant submits that as amended, each of claims 1, 14, and 18 is patentable over Morshed, such that all the claims rejected on this basis are patentable.

Independent claims 1, 14, and 18 have been amended as follows. The probe program is specifically "executed . . . by an interpreter." (See, e.g., patent application as filed, para. [0011]) The probe program is further specifically "interpreted by an interpreter," where this limitation originally was presented in claim 2, and has been removed from claim 2 and instead incorporated into all the independent claims. "[T]he interpreter run[s] on one or more processors." (See id.) The "probe program [is] independent of an architecture of the one or more processors." (See, e.g., patent application as filed, para. [0027]) Applicant submits that these claim limitations are not disclosed, suggested, or taught by or within Morshed. Specifically, Applicant submits that (1) Morshed does not disclose a probe program that is executed and interpreted by an interpreter, and (2) Morshed does not disclose a probe program independent of the architecture of the processors. Each of these claim limitations is now discussed in detail.

Probe program executed and interpreted by an interpreter

As noted, the independent claims have been specifically limited so that the probe program is "executed and interpreted by an interpreter." It appears that the Examiner has equated the various monitoring libraries (i.e., dynamically linked libraries, or DLL's) of Morshed as corresponding to the probe program of the claimed invention. For instance, the Examiner states that "[t]he library (probe program associated with each breakpoint) may include code that is invoked to gather various types of performance information." (Final office action, p. 5)

The Examiner has referenced column 41, line 43 of Morshed as indicating that the probe program of Morshed is executable *by* an interpreter. However, column 41, line 43 only states that "Java code . . . may be interpreted using the JavaVM" (i.e., virtual machine). It does not say that the probe program itself (i.e., the monitoring DLL) is interpreted.

The relied upon excerpt of Morshed in relevant part specifically states the following:

The techniques described . . . may be used, for example, with Java code as may be interpreted using the JavaVM. . . . . The client browser receives the page, *begins executing Java code* that may be included in the returned web page [where presumably this Java code is executed by an interpreter] *resulting in control being passed to the JavaVM DLL. The instrumented JavaVM byte code is executed and communicates with the monitor DLL* . . . . In one embodiment, control may be initially transferred to the monitor DLL.

(Col. 41, ll. 41-62) Note what is going on here. The Java VM can interpret Java code. However, the Java VM is described as communicating with the monitoring DLL, and indeed, initially transferring control to the monitoring DLL. Thus, the monitoring DLL – i.e., the probe program of Morshed – is not actually described as being interpreted or executed by an interpreter. The Java VM DLL interprets Java code, and can communicate with the monitoring DLL, and in fact can pass control to the monitoring DLL. In the latter instance especially, there is no way the Java VM DLL interprets the monitoring DLL, since it has passed control to the monitoring DLL.

Applicant respectfully submits that the Examiner is misconstruing what a dynamically linked library, or DLL, is. A DLL is a library of executable (not interpreted) code that a given computer program, such as one written in Java interpreted code, is linked to, so that, for instance, the program can run on a given platform. For example, as noted in the previous office action response, the web site http://www.techweb.com/encyclopedia/printArticlePage.jhtml?term=DLL defines DLL as an "executable program module . . . that performs one or more functions at runtime." DLL's are not interpreted; that is, they are not executed by an interpreter. Therefore, a DLL by definition is not interpreted or executed by an interpreter, such as a Java VM as in Morshed.

Applicant also has located another reference that informs what a DLL is. The "using native methods" of the "Java developer's guide" notes the following:

> In this chapter you'll learn how to access native C code from Java. You will see how to use the supplied tools to create a dynamic link library (DLL), which the Java runtime interpreter can *call* to perform native functions. When you finish this chapter, you will be able to link your Java code with native C methods.
>
> . . . .
>
> [K]eep in mind that native code will run only on the platform on which it was compiled. The fact that Java can call a function stored in a native DLL does not make the DLL more platform independent.

Thus, a DLL is *called by an interpreter*, specifically to perform native functions. A DLL is not *interpreted or executed by* the interpreter. Rather, the interpreter *calls* the DLL, and the DLL is executed – but the DLL is not executed *by the interpreter* nor is the DLL interpreted by the interpreter.

The Examiner has also quoted columns 19 through 23 of Morshed in detail on page 3 of the final office action. It is important to note that within this extensive excerpting of Morshed, the DLL's of Morshed that the Examiner equates with the probe program of the claimed invention appear only twice. First, "the class instance is provided to an instrumentation DLL 410." Second, "the instrumented class instance 412 contains native calls to a monitoring DLL 414." However, nowhere in this excerpt of Morshed, nor anywhere else in Morshed, is it said that the DLL's are executed or interpreted by an interpreter. That is, that a class instance is provided to a DLL, and contains native calls to another DLL, does not mean that the DLL's are executed or interpreted by an interpreter. Indeed, as has been noted above in relation to column 41, lines 41-62 of Morshed, control passes from the JavaVM interpreter to a monitor DLL. Thus, the byte code being interpreted by the JavaVM interpreter may contain a call to a DLL. As the JavaVM interpreter interprets this byte code, it will then call the DLL in question. The JavaVM does not execute or interpret the DLL itself.

Therefore, Applicant respectfully submits that Morshed does not anticipate the claimed invention. To anticipate a claim, the prior art reference must disclose each element of the claimed invention "arranged as in the claim" in question. (Lindermann Maschinenfabrik GmbH v. American Hoist & Derrick Co., 221 USPQ 481, 485 (Fed. Cir. 1984)) In the present situation, the Examiner found in Morshed a probe program – the monitoring DLL – and also where code is interpreted by an interpreter. However, in the claimed invention, it is the probe program that is interpreted and executed by an interpreter. This *arrangement* of the probe program being interpreted and executed by an interpreter is not disclosed in Morshed. Rather, Morshed discloses a probe program, and does disclose interpretation of code by an interpreter, but does not specifically disclose the probe program itself being interpreted by an interpreter. Indeed, Morshed discloses that the probe program is a DLL, which is by definition not executed or interpreted by an interpreter.

Probe program independent of the architecture of the processors

The Examiner interpreted the claims previously presented as requiring that the interpreter is independent of the architecture of the one or more processors. Applicant has specifically limited the claims so that it is the probe program that is independent of the architecture of the processors. (Applicant also notes as an aside that an interpreter cannot indeed by definition be processor architecture independent, since the point of an interpreter is to interpret and thus run a program on a particular architecture!) Applicant now discusses why Morshed does not disclose that its probe program is independent of the architecture of the processors.

The Examiner has referenced column 19, line 40 of Morshed as disclosing the processor architecture independence limitation of the claimed invention. Column 19, line 40 states that "[b]yte code is a format that is usually independent of the source language from which it was compiled, and which is intended to be independent of any computer hardware and/or operating system on which it might run." (Col. 19, ll. 39-43) Morshed goes on to say that "[b]yte code programs are executed by a software program sometimes referred to as a virtual machine." (Col. 19, ll. 43-45)

In other words, column 19, lines 39-45 of Morshed do not say that the *probe program* itself is byte code that is architecture independent, as is the probe program of the claimed invention. Rather, Morshed's discussion here just says what byte code is. The fact that byte code is independent of hardware, such as processors, does not mean that Morshed's probe program – its monitoring DLL and/or its other DLL's – is byte code. Indeed, a DLL is by definition not byte code, because byte code is interpreted by an interpreter, and a DLL is not interpreted by an interpreter, but rather is called by the interpreter. As stated above, the Federal Circuit in Lindermann has stated that a prior art reference must disclose each element of the claimed invention "arranged as in the claim." In the claimed invention, it is the probe program that is independent of the architecture of the processors. However, Morshed does not disclose this. It discloses a probe program, and discloses byte code that is architecture independent. Morshed does not disclose, in other words, the probe program itself being byte code that is architecture independent.

In fact, a DLL is by definition not architecture independent. As has been discussed above in relation to the Java developer's guide, a DLL is called by an interpreter to perform *native functions*. A native function is just that – a function that is native to a given processor architecture. As such, a DLL is the opposite of being architecture independent; it is architecture *dependent*. Something that performs native functions inherently cannot be architecture independent, since it has to be tied to that architecture (i.e., it has to be architecture dependent) in order to perform functions that are native to a given architecture. For all of these reasons, therefore, Morshed does not disclose a probe program that is architecture independent.

Conclusion

Applicants have made a diligent effort to place the pending claims in condition for allowance, and request that they so be allowed. However, should there remain unresolved issues that require adverse action, it is respectfully requested that the Examiner telephone Mike Dryja, Applicants' Attorney, at 425-427-5094, so that such issues may be resolved as expeditiously as possible. For these reasons, this application is now considered to be in condition for allowance and such action is earnestly solicited.

Respectfully Submitted,

[illegible]

August 30, 2007
Date

Michael A. Dryja, Reg. No. 39,662
Attorney/Agent for Applicant(s)

Law Offices of Michael Dryja
1474 N Cooper Rd #105-248
Gilbert, AZ 85233
tel: 425-427-5094
fax: 425-563-2098